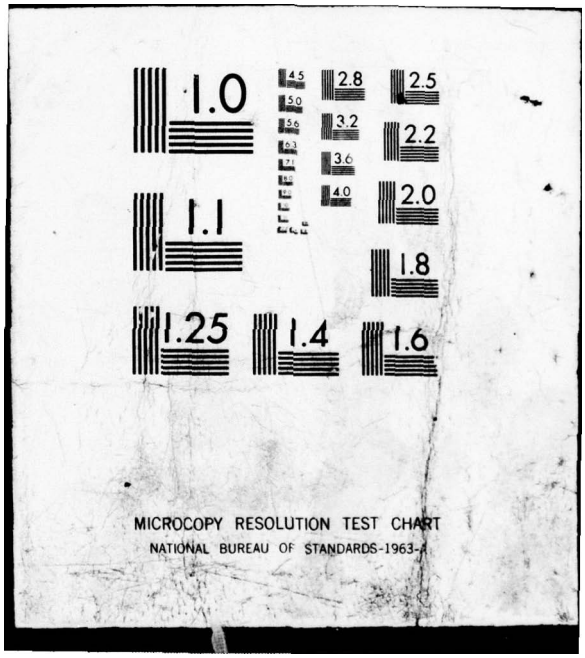
AD-A050 190 BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS F/G 9/2
DISTRIBUTED COMPUTATION AND TENEX-RELATED ACTIVITIES.(U)
JAN 78 R SCHANTZ, R THOMAS N00014-75-C-0773
UNCLASSIFIED BBN-3751 NL

1 OF 1
AD
A050 190

END
DATE
FILMED
3-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Bolt Beranek and Newman Inc.

Report No. 3751

Distributed Computation and TENEX-Related Activities

Quarterly Progress Report No. 10, 1 February 1977 to 30 April 1977

January 1978

Prepared for:
Defense Advanced Research Projects Agency

BBN Report No. 3751

DISTRIBUTED COMPUTATION AND TENEX-RELATED ACTIVITIES

Quarterly Progress Report No. 10

1 February 1977 to 30 April 1977

January 1978

Sponsored by:

Defense Advanced Research Projects Agency
ARPA Order No. 2935

Monitored by:

Office of Naval Research
Under Contract No. N00014-75-C-0773
Contract Period 1 November 1974 to 1 May 1978

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: BBN-[redacted]-3751	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): DISTRIBUTED COMPUTATION AND TENEX-RELATED ACTIVITIES.	5. TYPE OF REPORT & PERIOD COVERED: Quarterly progress rept. no. 10, 1 Feb - 30 Apr 77
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): R. Schantz, R. Thomas	8. CONTRACT OR GRANT NUMBER(s): N[redacted]14-75-C-[redacted]773
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, Massachusetts 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: Jan[redacted]78
	13. NUMBER OF PAGES: 43 (12) 47p
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2935.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

distributed computation
National Software Works
distributed operating system
TENEX operating system

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes ~~BBN~~ efforts in the design of the National Software Works system and ~~BBN~~ efforts to integrate TENEX into the National Software Works system.

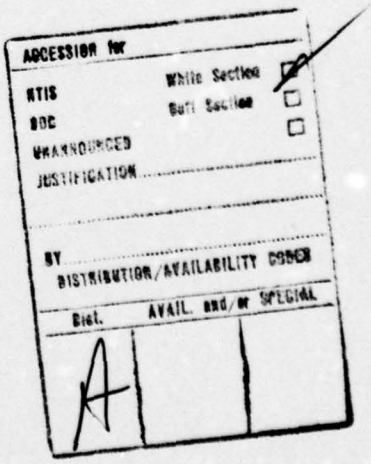
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060 100

TABLE OF COMTENTS



1. Introduction

The National Software Works (NSW) system development activities which we report for this quarter clearly indicate the substantial progress in the overall NSW effort. For the first time since the start of the project, a major portion of our activities have been diverted from developing a demonstrable minimum NSW logical functionality to tasks associated with providing an operational entity. The increased concern for both the reliability of the system, and for its overall performance under realistic conditions are a direct result of the maturing of the system functionality to a point where an operational NSW could be feasible. The extensive discussions of system performance and reliability in this progress report reflect this subtle change in project direction. Although there are still many problems yet to be attacked, the fact that the system has developed to the point where reliability and performance are important is indeed encouraging.

In the following sections we discuss the major accomplishments of this quarter. As usual, the bulk of the effort surrounds two major Tool Bearing Host (TBH) components, MSG and the Foreman. In addition to performance and reliability related developments, we also report on maintenance activities and some increased functionality in these components. Other activities reported include the distribution of additional NSW

documentation, and our participation in various meetings held with a number of groups to further the goals of the NSW project.

2. TENEX MSG Related Activities

During this quarter, the major part of our MSG related effort was spent in gathering and interpreting performance data for the TENEX MSG component itself, and for obtaining initial performance measures for the other NSW components as seen from within MSG. This work is discussed in detail in Section 2.2 and Appendix A. We have also released a sequence of new versions of the TENEX MSG component. These releases incorporate added functionality and provide solutions to problems that arose out of the increased use of MSG for NSW development activities. The nature of the changes are detailed in Section 2.1.

Another important part of this quarter's effort was the development and distribution of a TENEX MSG User Manual. This document (BBN Report No. 3540) describes from a user's point of view the TENEX implementation of MSG. It is intended as a reference for programmers who use MSG and as a guide for those responsible for the operation of systems, such as NSW, which use MSG. The user guide is the TENEX-specific companion document to the MSG design specification ("MSG: The Interprocess Communication Facility for the NSW," BBN Report No. 3483). The manual contains sections describing the MSG supported process interface (programmers guide) and descriptions of the various ways to define and run MSG configurations. Additionally, there is a section describing the use of the MSG process monitoring and

control functions, as well as an appendix enumerating the assigned error code values. The TENEX MSG User Manual is an important reference document for all MSG related activities, and will certainly contribute to the self-sufficiency of the various contractors using MSG. We intend to periodically update the manual in order to keep abreast of MSG implementation changes and the pending conversion of MSG for TOPS-20.

2.1 MSG Implementation Developments

There were a number of releases of new versions of TENEX MSG to the NSW project community during this reporting period. Since the communication subsystem is an integral part of almost all NSW component testing, we are especially sensitive to the need for prompt attention to any MSG software problems which arise from operational use. In an effort to expedite overall system testing, we will most often try to correct MSG bugs immediately as they are reported to us. When problems require more than a simple on-line modification, we usually immediately begin the process of releasing a new version of the software incorporating the more extensive changes necessary to correct the problem. In addition to the maintenance related changes, there were two other substantial modifications incorporated into MSG this quarter. These changes accommodate added functionality regarding the use of MSG by network dispatched NSW FE processes, and a substantial reorganization of the parts of MSG which handle direct connections.

The MSG job to handle network dispatched NSW FE processes is a little different from most other NSW MSG jobs. The job hierarchy for these FE jobs includes a Dispatcher related root node (responsible for managing the ARPANET Telnet connection to the user terminal) which is superior to the MSG task controlling process, which is superior to the FE process(es). The root node is tasked with ensuring the integrity of the job within the TENEX operating system, while the MSG process maintains the integrity of the MSG operation relative to the other co-located MSG jobs. It is the responsibility of the FE process to cooperate with other network-wide NSW components to ensure the integrity of the NSW system as a whole.

For an FE process to complete a normal termination, (e.g., after a user NSW LOGOUT), the FE must execute its termination and cleanup routines before relinquishing control. Likewise, the MSG process, upon detecting the termination of the FE, does its cleanup before it signals the root node to proceed with the job cleanup and deallocation. All of the preceding is the usual order of occurrence. However, there are circumstances in which the signalling is initially reversed. In particular, it is the root node that detects a break in the job's Telnet connection. This signal is then passed to the MSG process which until recently would immediately proceed with its termination sequence. However, new requirements for NSW functional behavior now give FE

processes a role in handling these conditions and mandate that the FE proceed with an "autologout" scenario with other NSW components. The autologout is used to properly save some of the user's context and to adjust system status tables so the user can continue at a later time. In order to carry out its protocol obligations when the user connection has failed, the FE must be informed of the failure and allowed sufficient time and resources to complete its operations. Accordingly, we have added such a capability to the TENEX MSG. Now, whenever an MSG for dispatched FE jobs is signalled about a broken Telnet connection, MSG, in turn, signals the FE process via a channel previously set up by the FE. MSG then allows the FE to complete its cleanup before proceeding with the job deallocation sequence.

An extensive effort was also made this quarter to improve the code for handling direct ARPANET connections within MSG. Because of the asynchronous, multi-step procedures required for both opening and closing ARPANET connections, the MSG code to handle these functions is complex. It is implemented as a finite state machine, but this simple model of operation becomes somewhat obscured by the requirement for a large number of possible states along with a large number of possible transitions. Due to its complexity and its inherent asynchronous nature, extensive testing and debugging is very difficult. Because of this, many of the problems with direct connections

have come to light only after substantial field use of MSG. The types of failures reported, as well as the opportunity to "catch" and examine transient failures as they occurred in actual use, led to an increased understanding of the measures which were necessary to prevent certain error producing situations from occurring and in successfully dealing with others when they do occur. The effort expended in modifying the direct connection handling in MSG should make subsequent releases of MSG much more reliable and well behaved in this area.

2.2 MSG Performance Measurements

As reported last quarter, we have developed and installed an instrumentation package for the TENEX MSG component. The data which is collected can be used to measure the overall performance of the MSG component in delivering messages, sending alarms, and setting up network connections. In addition, by recording selected process parameters whenever a process interacts with MSG, measurements can be obtained to estimate certain aspects of NSW component behavior in carrying out the various NSW protocol scenarios. We have now performed a number of experiments to obtain both types of performance data and have written a fairly extensive report summarizing our findings. The purpose of the measurement experiments was to gain a quantitative understanding of the operation of the NSW system in order to identify performance bottlenecks and to determine ways to improve NSW

performance. Part of our objective was also to develop support software and methodologies to be used for subsequent, more detailed measurements.

Two distinct sets of measurements were taken. The first class of measurements were for MSG operations, and measured the elapsed real time for performing these operations under various conditions. The second class of measurements were for the various higher level NSW components, (e.g., Front End, Works Manager, etc.), and measured the CPU requirements for each under the various NSW functional operations, (e.g., RUNTOOL, OPEN-FILE, etc.). The performance measurement summary, including sections on the measurement methodology as well as the observed results, is included in as Appendix A of this progress report.

2.3 An MSG Performance Experiment

In an effort to improve MSG's response characteristics, we have experimented with a change in the layout of MSG internal data structures. The thought behind the experiment was to see if a different layout might minimize the number of data pages that need to be accessed in completing an operation. This, in turn, might minimize the number of page faults, which might then contribute to decreasing the delay associated with the various MSG operations.

Currently, most MSG data structures, (i.e., tables for message block entries, connection blocks, etc.) are organized as a series of parallel tables. That is, each data field of a given n-word entry is retrievable as the same index (i) into each of n different single dimension arrays. Put another way, the data structure representing the ith item is the composite of the ith entry for each of the n parallel tables. We call this a horizontal data structure. We can contrast this to another organization technique in which the n-words of each entry are stored in consecutive locations, with the i+1st item in its entirety following the ith. We call this a vertical data structure. While both organizations accommodate the same data, one or the other may be more appropriate depending upon the access patterns to the data. For example, if we are primarily dealing only with a specific field of an entry in conjunction with the same field of other entries, then a horizontal table might be appropriate. If, on the other hand, accessing one data field of an entry normally also means an immediate reference to the other fields of the same entry, then a vertical structure might be more appropriate.

At the outset in coding the MSG component, we developed and utilized a series of macro functions which are used to define and access internal data structures. Because of this, it was a relatively easy task to convert to an alternative table

organization. We performed the experimental conversion and ran a number of tests. The results were that by utilizing a vertical table organization we were able to achieve about a 20-25% improvement in the "paging" characteristics of MSG over using a horizontal structure. The improvement relates to the measurement of the working set size, the number of page faults, and the average CPU time per page fault to do equivalent tasks under roughly similar conditions. However, we also found that despite the improved locality of reference exhibited by MSG, the delay associated with the various MSG operations did not improve. Thus, we conclude that MSG performance is not currently being limited by its paging behavior. Other experimentation is continuing.

3. TENEX Foreman Related Activities

This quarter there were a number of new releases of the TENEX Foreman component. These releases corrected various problems that arose in testing and component integration by other contractors, as well as from limited use by outside organizations such as the system development group at NELC. In addition, there were major implementation efforts in the areas of reliability and instrumentation. A few new features have also been added to the Foreman, and another tool (the text formatting program MRUNOFF) has successfully been added to the available NSW repertoire. These developments are elaborated further in the following subsections.

3.1 Implementation to Support the Interim Reliability Plan

Last quarter we reported on the design effort to incorporate NSW file system reliability concepts into the existing NSW component structure (see BBN Report No. 3736, Section 3.1). This quarter we can report that the so called "interim reliability plan" has been finalized, and also that its implementation for the TENEX Foreman component is complete. As we have previously noted, the plan is an attempt to provide roughly the same degree of safety regarding losing files and completed user work as could be expected by a user dealing directly with the tool bearing host operating system. The problem addressed by the reliability plan

is brought about by one of the primary design goals of NSW. That is, the user must not deal directly with any of the constituent TBH operating systems. Rather, in order to incorporate a single, uniform view of the entire system, the user is permitted to interact only with NSW which intercedes on his behalf with the local operating system. Thus, we require mechanisms within NSW itself to assure adequate safety in handling the NSW file catalog and the files themselves whether they currently reside in the tool workspace or in the NSW file space.

The interim reliability plan requires each Foreman, during a tool session, maintain a local name dictionary (LND) on a non-volatile storage medium to allow file retrieval in the event of a system failure. The LND is a temporary local file directory that contains such information as the user's name for an NSW file, and the local host name for the image of that file. The LND for a normally terminating tool session is not discarded until the WM "guarantees" the acceptance and "permanent" recording of the appropriate workspace files. Any session which does not terminate normally (that is, with the successful completion of file delivery and cleanup operations) has its LND automatically saved. Saved LNDs are subsequently reported to the WM so that workspace files which have not yet been incorporated into the NSW file system can later be retrieved by the affected user.

The TENEX Foreman has been extensively modified to implement these reliability related concepts, as well as to incorporate the spirit of these changes in other reliability related areas not specifically addressed by the interim plan. On TENEX, the LND is maintained as part of the TENEX file system to ensure its existence across system restarts. Convenient access to the LND data base is achieved by "mapping" the LND file pages into the virtual address space of the appropriate Foreman process. When a tool session is begun, the Foreman selects a workspace from the pool of available directories for handling the file storage requirements for the session. The LND is itself a file resident in the workspace while it is active.

Most error conditions detected by the Foreman during a tool session (e.g., message timeouts) cause the tool session to be saved. In addition, the FE can request that the Foreman save a tool session in the event that the FE loses contact with its user (see also the discussion of autologout in Section 2.1 of this report). Tool sessions which abruptly come to a halt because of a host system "crash" are automatically saved when the NSW is next restarted on the effected TBH. The implementation of the saving, reporting and rerunning of the tool sessions is centered around two shared Foreman data bases. (Each TENEX TBH has its own private copy of the data bases, which are accessible to all Foremen on the TBH.) One data base describes the state of all

accessible NSW workspaces (i.e., the workspace list) and the other companion data base contains the state of all saved tool sessions (i.e., the rerun list). Saved tool sessions are currently maintained in the workspace in which they were originally run. This means that part of the procedure for saving a tool session is ensuring that the workspace which supported the session is not placed back into the pool of available spaces when the Foreman is deallocated. This allows another Foreman at some subsequent time to complete ("rerun") the session merely by connecting to the appropriate workspace which already has the appropriate LND and supporting files. However, it also means that the workspace is not available for running other tools while the session is still saved. For the initial implementation we evaluated this as an acceptable tradeoff.

Whenever a tool session is saved, the appropriate workspace data base entry is marked as containing a saved session (to prevent it from bing reused) and a rerun list entry for the session is created. The rerun list entry will be used to coordinate the activities which are appropriate for saved tool sessions (e.g., reporting them to the WM, rerunning). The creation of the rerun list entry along with the appropriate exchanges with the WM to report the saved session can be initiated in two different ways. If the Foreman process controlling a tool is alerted to or detects some error condition,

it adjusts the data bases to reflect the new status and then immediately tries to report the saved session. For those cases in which the tool Foreman does not retain control (e.g., system or Foreman crash), a special Foreman process will save and report all previously unreported sessions as a group, when the TBH software is next reinitialized. After a saved tool session is properly recorded in the FM data base and reported to the WM, it can at any time be rerun. The rerun request is a generic message to a Foreman which then searches the rerun list for the saved session. After a successful search, the Foreman can locate and access the saved LND and its associated files and properly terminate the session.

We have completed the initial implementation in accordance with the specifications set forth in the interim reliability plan. We have also tested these changes in stand alone fashion to the degree possible without having the supporting matched components. Other contractors are now finishing the implementation of their component responsibilities under the plan, and we are beginning to mount a large scale system integration effort for all of the updated components.

3.2 Foreman Instrumentation

Another major implementation effort undertaken this quarter was in the area of instrumentation for the Foreman. We have

designed and implemented a package handling three different types of instrumentation needs within the Foreman. The overall approach is to use probes inserted throughout the Foreman corresponding to specific events. Each time a probe is executed, selected information is placed in a core buffer marking the event, time of occurrence, and other raw data relating to the nature of the event. Immediately before asking to be deallocated, the individual Foreman process dumps the contents of the event recording buffer along with appropriate header information into a common file shared with other Foreman processes. These files can then be processed off-line. This approach has been taken to minimize the overhead for handling the instrumentation while processing user demands.

There are three basic uses for the probes, with use each being configured differently. One probe type is used for recording message traffic both to and from the Foreman process. This is very helpful when attempting to reconstruct the sequence of events leading to a particular internal process state. It has also proven useful in pinpointing the origin of protocol violations in transmitted messages.

A feature of the FM instrumentation package is the ability to have the entries being recorded in the "core buffer" converted to text and displayed on the job controlling terminal as they are recorded. This is quite useful as an on-line testing and

debugging aid, although it can introduce substantial delay in servicing messages and would invalidate most performance measurement recording. Certain probes are inserted mostly for their tracing effect to be used in an on-line terminal environment.

A second type of probe is used to report unusual events and error conditions occurring in an unattended system. One mode of FM operation is to have an individual process halt on detecting an inconsistent situation. This is useful in trying to uncover the problem but is not feasible unless the system is monitored by someone thoroughly familiar with the operation of the FM. Since this is not normally the case, we usually have the FM set up to clean up as best it can on error conditions, then try to gracefully terminate its operation. The trouble here is that the cause of the error often goes undetected. To remedy this, we have utilized special probes which try to capture some of the relevant state of the FM when an error condition is detected. These probes insert data into the log file which can be used off-line to not only detect the occurrence and frequency of failures, but also to attempt to reconstruct their cause. We have already discovered and fixed a number of FM bugs which came to our attention days after the errors actually occurred in NSW configurations managed by other contractors.

The third type of probe is used to obtain performance measurements. These probes record process characteristics such as CPU time used and connect time, along with more global characteristics such as system and group load averages. Measurement probes are generally used in pairs to obtain time intervals (CPU time and/or real time) for certain events. To gain experience with the measurement technique and in trying to formulate a measurement methodology, we will experiment with measuring both internal FM performance for selected large grain events (e.g., BEGINTOOL) and external NSW performance in handling FM related functions (e.g., time to complete an Open-file request).

The probe handling software is currently undergoing checkout. When complete, a new FM will be distributed with probes in place to automatically record error conditions and performance data.

3.3 Other Implementation Developments

A number of operational problems were corrected this quarter through a series of FM system component releases to the NSW community. The most notable fix corrects the problems associated with race conditions arising out of "simultaneous" FE and FM instigation of similar functions in trying to terminate the tool. On certain occasions, these races caused some files to be

delivered twice when a user tried to terminate a tool both from within the tool and also via the NSW command language. A similar problem caused occasional abort commands to be ignored when received after a tool had begun its termination sequence. To correct these problems we have transformed the existing tool status descriptor into a multiple component state variable with finer grain states which can now adequately depict all of the various combinations of legal termination sequences. The message handling code is now integrated with the complete tool status variable to achieve a proper functionality.

New releases of the Foreman also now support the STOP-TOOL and START-TOOL (SUSPEND, RESUME) features as described in the Foreman Specification document. In theory, a user can now instruct his FE, via the NSW command language, to suspend an active tool, and later to resume its execution. In practice, the NSW FE has not implemented the support for these functions so they are not as yet part of the primitive tool control set.

4. Documentation and Meetings

This quarter we have issued an update to the Tool Builder's Guide manual (Massachusetts Computer Associates, Document CADD-7702-1811, BBN Report No. 3308. This update reflects the wide ranging changes in the NSW over the last year. The description of both NSW characteristics relating to the tool interface and the operational procedures for inserting a tool in NSW have been revised to reflect the current system release. Also in the documentation area, we have issued an updated Appendix to the Foreman specification document detailing the messages and patterns of use for implementing the interim reliability scenarios. In a break with the past, all messages dealing with the reliability constructs for all components are now compiled within a single document.

There were a number of inter-contractor meetings this period to define and then to coordinate the implementation of the interim reliability measures. We have also worked closely with personnel from Massachusetts Computer Associates in designing the NSW measures to be adopted within the context of a larger, more extensive reliability plan. Toward that end, we were meeting on a weekly basis to discuss these technical issues.

Finally, at the request of the ARPA office, we met with Professor Howard Morgan of the University of Pennsylvania. We

discussed the possible use of the MSG facility for supporting a data base alerting service which he is currently working on. MSG's message oriented communication approach seems well suited to this type of data base processing system, and we expect the dialog to continue.

Appendix A: NSW Performance Measurement Report

The following is a copy of the report that was circulated to NSW project members in April. It summarizes the results of a fairly extensive effort to organize a set of experiments which capture performance data for the various NSW components.

Preliminary NSW Performance Measurements

Performed by BBN

April 1977

1. Introduction.

This report summarizes a set of performance measurements made on the NSW system. The purpose of these measurements and ones to follow is to gain a quantitative understanding of the operation of the NSW system. With this understanding, it should be possible to identify performance bottlenecks and to determine how to improve NSW performance.

The measurements reported here should be regarded as the first set of a series of measurements to be performed on NSW. These measurements were largely exploratory in nature. The primary objectives were twofold: to derive a general quantitative feeling for NSW operation in order to identify aspects of system operation to be more thoroughly measured; and, to develop software and methodologies to be used for subsequent, more detailed measurements.

Two distinct classes of measurements were performed. The first class of measurements were for MSG, the NSW interprocess communication facility. The delay (elapsed real time) incurred to perform various MSG primitives was measured both for intra-host and inter-host operation. These measurements are reported in Section 2. The second class of measurements were for the various NSW components (e.g., Front End, Works Manager, File Package, Foremen). The CPU time required by each of the components to perform its part of an operation was measured for various operations (e.g., RUNTOOL, OPEN, DELETE, etc.). These measurements are reported in Section 3.

Note that in one case (for MSG) we have measured delay and in the other case (for NSW components) we have measured CPU requirements. For these preliminary measurements these different properties for the two cases were chosen because we believed them to be both the easiest to measure and the most interesting in each case. Subsequent experiments will be made to measure MSG CPU requirements and NSW component delays for various scenarios.

As noted above, the measurements reported here are preliminary. In this report we have refrained, as much as possible, from interpreting the measurement data. We have, however, tried in a few cases to make comments of an explanatory nature. We believe that these measurements are likely to raise more questions than they answer. We believe that this is good and that the answers to these questions will come both from more comprehensive and directed measurements and from re-examination of component implementatons and inter-component protocols. We expect to work closely with other NSW project personnel to develop a plan for additional performance measurements and subsequent performance improvement efforts.

2. Measurements of MSG.

2.1 Measurement Method.

A series of experiments were run to measure delays incurred performing various MSG primitive operations. The measurements were made using two processes, called M1 and M2, which cooperate to exchange messages, send alarms, and open and close connections. Round-trip delays are directly measured from which one-way delays can be derived.

Using M1 and M2, measurements can be made for the SendGeneric, SendSpecific, SendAlarm, OpenConn and CloseConn primitives. The SendGeneric experiment measures the elapsed time for M1 to send a generic message of N bytes to M2 and to receive a reply (specific) message of N bytes from M2. The SendSpecific experiment is similar with the exception that the message from M1 is a specific message. The SendAlarm experiment measures the elapsed time for M1 to send M2 an alarm and to receive a reply alarm from M2. For the connection experiments, M1 and M2 establish a connection of a given type, exchange some data, and then close the connection. The delay to open the connection and the delay to close the connection are measured.

The M1 process accepts a specification of the measurements to be made from a user, ininteracts with the M2 process to inform it of the measurements to be made, and then cooperates with the M2 process to perform the measurements. As an example, consider an experiment to measure SendSpecific delays by sending 1000 400-byte messages. First, M1 would send M2 a (generically addressed) message specifying that 1000 400-byte messages were to be exchanged. Next, M1 would measure the delay incurred in sending a 400-byte message to M2 and receiving a 400-byte reply message from M2. After 1000 repetitions of the above message exchange, M2 would terminate and M1 would produce a summary of the experiment. The summary includes: the average, maximum and minimum measured delays; histograms of measured delays; and histograms of system and group load averages which are measured by M1 each time an MSG primitive is executed. Appendix 1 is a summary produced for a typical experiment.

A detailed specification of the M1 and M2 processes given in a stylized programming language appears in Appendix 2. We recommend that M1 and M2 processes be implemented for all MSG hosts so that MSG performance measurements can be made for all combinations of MSG host types.

2.2 Measurement Data.

SendGeneric delays:

```
        M1:             M2:
        SendGeneric ->
                    <- SendSpecific
```

100 Repetitions for each case:

Host M1	M2	Message Size (bytes)	Measured Delay (ms) Avg	Max	Min	1-way Delay (ms)*
BBNB	BBNB	50	727	2611	366	469
"	"	100	653	3682	363	337
"	"	200	732	4279	357	469
"	"	400	813	4322	361	565
BBNB	BBN	50	1269	9025	638	580
"	"	100	1337	4145	725	786
"	"	200	1463	8603	753	874
"	"	400	1434	13057	805	688
BBN	SRI-KA	50	2062	6331	1352	1125
"	"	100	3036	16455	2273	1660
"	"	200	3139	5945	2327	1706
"	"	400	3545	6388	2547	1969

* (Avg measured delay) - (derived 1-way SendSpecific delay)

SendSpecific delays:

```
        M1:             M2:
        SendSpecific ->
                    <- SendSpecific
```

100 Repetitions for each case:

Host M1	M2	Message Size (bytes)	Measured Delay (ms) Avg	Max	Min	1-way Delay (ms)*
BBNB	BBNB	50	516	1088	345	258
"	"	100	632	2372	393	316
"	"	200	526	1342	351	263
"	"	400	496	972	345	248
"	"	800	527	1960	349	264
BBNB	BBN	50	1377	9768	627	689
"	"	100	1101	4975	725	551
"	"	200	1178	3382	787	589
"	"	400	1492	14545	964	746
"	"	800	2540	27031	1078	1270

BBN	SRI-KA	50	1933	3593	1425	937
"	"	100	2752	5032	2062	1376
"	"	200	2865	4884	2113	1433
"	"	400	3152	6204	2391	1576
"	"	800	3498	4889	2822	1749

* (Avg measured delay) / 2

SendAlarm delays:

```
        M1:           M2:
        SendAlarm ->
                  <- SendAlarm
```

200 Repetions for each case:

Host M1	M2	Measured Delay (ms) Avg	Max	Min	1-way Delay (ms)*
BBNB	BBNB	361	2075	261	181
BBNB	BBN	1110	16345	473	555
BBN	SRI-KA	1630	3941	1128	815

* (Avg measured delay) / 2

Connection measurements:

```
        M1:                M2:
        OpenConn       <-> OpenConn
        Send 100 bytes  ->
        CloseConn      <-> CloseConn
```

100 Repetitions for each case:

OpenConn delays:

Host M1	M2	Conn Type	Measured Delay (ms) Avg	Max	Min
BBNB	BBNB	Bin Send	929	4715	451
"	"	Bin Pair	1029	4318	607
BBNB	BBN	Bin Send	1278	8683	332
"	"	Bin Pair	1615	6096	488
BBN	SRI-KA	Bin Send	1161	2837	721
"	"	Bin Pair	1358	3341	783

CloseConn delays:

Host M1	Host M2	Conn Type	Measured Delay (ms) Avg	Max	Min
BBNB	BBNB	Bin Send	814	3297	471
"	"	Bin Pair	825	4437	494
BBNB	BBN	Bin Send	1398	7766	336
"	"	Bin Pair	1276	4794	421
BBN	SRI-KA	Bin Send	888	2018	591
"	"	Bin Pair	1068	1944	642

2.3 Comments.

1. All measurements were made under very light TENEX loads. Consequently, we believe that the "Min" figures approximate lower bounds on delays for the operations measured (for the current TENEX MSG inplementation).

2. Intra-host message delays are independent of message length. This is to be expected since intra-host messages are handled by TENEX page mapping operations which are independent of the size of the messages within a page. Inter-host message delays, in general, increase with message size as might be expected since the message must be sent as a stream of bytes across the network.

3. The SendGeneric measurements were made for an M2 process that executes a StopMe primitive after its SendSpecific reply to M1. The "termination specification" for M2 was "RESTARTPROC" (see TENEX MSG User Manual). Subsequent measurements should be made to determine how, if at all, delay depends upon the termination specification for M2. In addition, measurements in which M2 executes a new ReceiveGeneric rather than a StopMe should be compared with the above measurements.

4. We can not explain the large maximum delays for some experiments. For example, we believe that the response of the TENEX scheduler to transient load variations does not adequately account for the single delay of 13.057 seconds for the 400-byte intra-host SendGenerics (without which the average delay would be 8% less). The cause of these gross delays should be investigated.

3. Measurement of TENEX NSW Components.

3.1 Measurement Method.

A series of experiments were conducted to measure the CPU time required by NSW components to perform standard NSW operations. The measurements were made using the event logging capability of MSG which, when enabled, logs the occurrence of various "interesting" events. Appendix 3 contains a small portion from the MSG event log for one of the experiments along with a description of the various log entries.

The general experimental procedure was fairly simple. An NSW configuration was initialized with MSG event logging enabled and standard scenarios for user sessions were used to exercise the system. With a knowledge of the user scenarios and the underlying inter-component protocols, the resulting MSG event log was analyzed to determine component CPU times for the various NSW operations. All experiments were performed with only a single NSW user.

In general, these measurements reflect internal component CPU requirements exclusive of CPU requirements of MSG to support inter-component communication. For "servers" (e.g. Works Manager processes), the CPU quantity is generally measured from receipt of a request to completion of the request. For requesting components, CPU time is generally measured from initiation of a request until the request is satisfied; this measurement would, of course, include any activity by the component to participate in the protocol scenario, if any, associated with the request.

3.2 Measurement Data

The measurements were made on the BBNB TENEX for an NSW system usually used for debugging purposes. This was strictly a single host system; there were no inter-host interactions.

The following "certification" data serves to identify the components measured in these experiments:

Works Manager:
CERTIFICATION: BOLDUC,bbnb,<WMSRC>WM.SAV;107,
4-Feb-77 12:04:42

TENEX File Package:
CERTIFICATION: R. FANEUF,BBNB,<FANEUF>FLPKG.SAV;32,
11-Feb-77 10:57:36

TENEX Front End:
CERTIFICATION: bearisto,bbnb,<BEARISTO>FE.SAV;92,
7-Feb-77 13:51:55

TENEX Foreman:
CERTIFICATION: MASSEY,BBNB,<BBN-NSWTST>FOREMAN.SAV;1300,
8-Mar-77 16:14:00

Measurements were made for the RUNTOOL, TOOLHALT, OPEN, DELIVER, and DELETE operations. These operations were chosen as representative of NSW operations. Unless otherwse noted, all measurements were made under light TENEX loads.

RUNTOOL:

Protocol Scenario:

1.	SendGeneric	FE -> WM
2.	SendGeneric	WM -> FM
3.	SendSpecific	FM -> WM
4.	SendSpecific	FM -> FE
5.	OpenConn	FM -> FE
6.	OpenConn	FE -> FM
7.	SendSpecific	WM -> FE

CPU quantities measured (ms):

FE: Time from execution of SendGeneric (1) to execution of ReceiveSpecific following (7).

WMT: Time from receipt of generic message (1) to execution of ReceiveGeneric following (7).

WM1: Time from receipt of generic message (1) to execution of SendGeneric (2).

WM2: Time from receipt of specific message (3) to execution of SendSpecific (7).
(Note: WM1+WM2 will be somewhat less than WMT due to WM processing subsequent to SendSpecific (7)).

FM: Time from receipt of generic message (1) to completion of OpenConn (5).

TOTAL: FE+WMT+FM

ID	FE	WMT	WM1	WM2	FM	TOTAL
1	590	4787	2073	2649	1688	7045
2	428	4753	2152	2554	1172	6353
3	360	4604	2095	2490	1235	6199
4	410	4771	2089	2638	1426	6607
5	329	4551	2036	2481	1752	6306
6	457	4643	2044	2552	1220	6320
7	356	4600	2050	2539	1437	6393
8	348	4635	2078	2537	1928	6911
9	376	4582	2075	2487	1417	6375
10	375	4542	2056	2456	1192	6109
11	344	4639	2079	2536	956	5939
12	244	4473	2099	2349	865	5582
13	385	4538	2027	2473	924	5847
14	399	4595	2099	2487	1043	6037
15*	556	4950	2163	2726	1322	6828
16*	1127	5086	2332	2688	2433	8646
17*	746	5166	2233	2842	1150	7072
18*	981	5260	2307	2890	1316	7557
19*	1758	5775	2645	3039	1219	8752
20**	300	4798	2101	2669	1060	6158
21**	277	4652	2174	2470	1009	5938
22**	341	4854	2196	2624	1054	6249
AVG	522	4784	2146	2599	1309	6615

* Very high TENEX load; FM statistics package enabled.
** Low TENEX load; FM statistics package enabled.

TOOLHALT

Protocol Scenario:

1.	SendSpecific	FM -> FE
2.	SendGeneric	FM -> WM
3.	CloseConn	FE -> FM
4.	CloseConn	FM -> FE
5.	SendSpecific	WM -> FE
6.	SendSpecific	FE -> WM
7.	SendSpecific	WM -> FE
8.	SendSpecific	WM -> FM

Note: For all measurements below, the tool termination is initiated by the tool itself (e.g., TECO ";h", MACRO "^Z"). The embedded exchange of specific messages between the WM and FE, (5) and (6), is apparently used to report tool charges to the user.

CPU quantities measured (ms):

FE: Time from receipt of specific message (1) to execution of ReceiveSpecific primitive after (8).

WMT: Time from receipt of generic message (2) to execution of ReceiveGeneric after (8).

WM1: Time from receipt of generic message (2) to execution of SendSpecific (5).

WM2: Time from receipt of specific message (6) to execution of SendSpecific (7).

WM3: Time from execution of SendSpecific (7) to execution of SendSpecific (8).

FM: Time from execution of SendSpecific (1) to execution of StopMe after completion of CloseConn (4).

TOTAL: FE+WMT+FM.

ID	FE	WMT	WM1	WM2	WM3	FM	TOTAL
1	317	4789	3503	399	863	317	5423
2	547	4918	3514	477	902	306	5771
3	355	4743	3441	400	891	303	5411
4	333	4785	3466	435	876	315	5433
5	436	4854	3438	484	909	312	5602
6	365	5161	3995	343	812	307	5833
7	370	5566	4108	459	980	330	6266
8	278	5207	3940	408	852	299	5784
9	378	5365	3981	471	897	328	6071
10	327	5219	3902	414	894	300	5846
11	352	4825	3497	416	901	291	5468
12*	619	5273	3656	473	1074	639	6531
13*	965	5430	3802	525	1038	668	7063
14*	1049	5525	3852	529	1076	703	7277
15*	1019	5966	4298	513	1081	793	7776
16*	1527	6661	4691	625	1283	503	8691
17**	299	5001	3634	411	937	1053	6353
18**	345	7731	6327	449	945	1083	9159
AVG	549	5390	3947	457	956	492	6922

* Very high TENEX load; FM statistics package enabled.
** Low TENEX load; FM statistics package enabled.

OPEN

Protocol Scenario:

1.	SendGeneric	FM -> WM
2.	SendGeneric	WM -> FLPKG
3.	SendSpecific	FLPKG -> WM
4.	SendSpecific	WM -> FM

Note: OPEN is a local copy from NSW file storage space into tool workspace; this is the case because the NSW configuration being measured is a single host system. In all cases the file opened was very small (<50 bytes) and its name was unambiguously specified.

CPU quantities measured (ms):

WMT: Time from receipt of generic message (1) to execution of ReceiveGeneric following (4).

WM1: Time from receipt of geneic message (1) to execution of SendGeneric (2).

WM2: Time from receipt of specific message (3) to execution of SendSpecific (4).

FLPKG: Time from receipt of generic message (2) to execution of ReceiveGeneric after (3).

TOTAL: WMT+FLPKG

ID	WMT	WM1	WM2	FLPKG	TOTAL
1	5645	3988	1565	1716	7361
2	5615	3989	1607	1723	7338
3	5676	4056	1564	1797	7473
4	5709	4083	1605	1688	7397
5	5623	4002	1610	1848	7471
6*	6149	4321	1753	2088	8237
7*	5814	4065	1671	2023	7837
8	5480	3875	1596	2043	7523
9	5440	3882	1545	1989	7429
10	5506	3991	1505	1998	7504
11	5557	3997	1640	1986	7543
12	5304	3860	1505	2023	7327
13	5515	3887	1618	2044	7559
14	5661	4000	1643	2022	7683
AVG	5621	4000	1602	1928	7549

* Very high TENEX load.

DELIVER

Protocol Scenario:

1.	SendGeneric	FM -> WM
2.	SendGeneric	WM -> FLPKG
3.	SendSpecific	FLPKG -> WM
4.	SendSpecific	WM -> FM

Note: DELIVER, like OPEN, involves a local file copy. For DELIVER, the copy is from tool workspace into NSW file space. As in the OPEN measurement, the file in question was very short and the file name always unabiguously specified. In addition, all of the DELIVER operations created new files rather than replaced existing ones.

CPU quantities measured (ms):

WMT: Time from receipt of generic message (1) to execution of ReceiveGeneric following (4).

WM1: Time from receipt of geneic message (1) to execution of SendGeneric (2).

WM2: Time from receipt of specific message (3) to execution of SendSpecific (4).

FLPKG: Time from receipt of generic message (2) to execution of ReceiveGeneric after (3).

TOTAL: WMT+FLPKG

ID	WMT	WM1	WM2	FLPKG	TOTAL
1	4271	2797	1464	2613	6884
2	4231	2740	1463	2607	6838
3	4194	2735	1449	2672	6866
4	4161	2707	1447	2681	6842
5	4303	2775	1472	2612	6915
6*	4810	3181	1549	2302	7112
7	4336	2845	1474	2963	7299
8	4262	2746	1506	3064	7326
9	4285	2758	1519	2926	7211
10	4090	2690	1391	3001	7091
11	4135	2688	1437	3016	7151
12	4106	2840	1258	2934	7040
AVG	4265	2792	1452	2783	7048

* Very high TENEX load.

DELETE

Protocol Scenario:

1.	SendGeneric	FE -> WM
2.	SendSpecific	WM -> FE
3.	SendSpecific	FE -> WM
4.	SendGeneric	WM -> FLPKG
5.	SendSpecific	FLPKG -> WM
6.	SendSpecific	WM -> FE

Note: The file name was unambiguously specified in all cases. The embedded specific message exchange between WM and FE, (2) and (3), apparently is used to confirm that the user really wants to delete the file in question.

CPU quantities measured (ms):

FE: Time from execution of SendGeneric (1) to execution of ReceiveSpecific after (6).

WMT: Time from receipt of geneic message (1) to execution of ReceiveGeneric after (6).

WM1: Time from receipt of generic message (1) to execution of SendSpecific (3).

WM2: Time from receipt of specific message (3) to execution of SendGeneric (4).

WM3: Time from receipt of specific message (5) to execution of SendSpecific (6).

FLPKG: Time from receipt of generic message (4) to execution of ReceiveGeneric after (5).

TOTAL: FE+WMT+FLPKG

ID	FE	WMT	WM1	WM2	WM3	FLPKG	TOTAL
1	408	4158	1924	555	1651	777	5343
2	400	4083	1927	548	1588	682	5165
3	326	4001	1885	520	1557	843	5170
4	328	3946	1947	508	1481	777	5051
5	401	4133	1901	555	1652	819	5353
AVG	373	4064	1917	537	1586	780	5216

3.3 Comments.

1. For all the above measurements, the WM message-loging feature was enabled. Disabling WM message-logging can be expected to reduce WM CPU figures somewhat.

2. WM CPU figures have been broken down further than those for other components for two reasons:
   a. The WM role in all scenarios above generally consists of clearly identifiable sub-roles;
   b. We were somewhat surprised by the extensive WM CPU requirements and thought it would be useful to have a more detailed breakdown of those requirements.

3. To put the above figures into perspective, consider the OPEN scenario which (exclusive of MSG CPU requirements) requires 7.5 seconds of CPU time to open a short file. Roughly speaking, this means that a TENEX dedicated to NSW would be able to satisfy between 7 and 8 such file open requests in a minute, assuming the dedicated TENEX was involved in no other NSW activity.

4. The large variance in FE and FM CPU requirements in RUNTOOL and TOOLHALT requires further investigation.

5. On the surface of it, OPEN and DELIVER appear to be similar operations, differing primarily in the direction of the file movement between NSW filespace and tool workspace. It is, therefore, somewhat surprising that WM and FLPKG CPU times are not approximately the same for these operations. Investigation of the operation of these components with respect to OPEN and DELIVER seems to be indicated.

Appendix 1 - Sample MSG Measurement Summary

```
MSG Measurements: Local Specific - Size = 50

Started: 13-Apr-77 20:42:37
Stopped: 13-Apr-77 20:43:32
Source Host: BBN-TENEXB
Target Host: BBN-TENEXB
Round trip delay for:
        SendSpecific ->
                     <- SendSpecific
100 Repetitions
Message Size = 50 (8 bit bytes)
Start Load Averages (Group/System) =   3.19/3.08
End Load Averages (Group/System) =   5.01/4.33
Average delay (ms): 516.03  Min/Max delays: 345/1088
Bin        No.
64         0
128        0
256        0
512        67
1024       32
2048       1
4096       0
8192       0
16384      0
32768      0
>          0

Load Histograms:

System 1 Minute Loads:

Bin        No.
  1.00     0
  2.00     0
  3.00     0
  4.00     125
  5.00     77
  7.00     0
  9.00     0
 11.00     0
 14.00     0
 17.00     0
>          0
```

Group Loads:

Bin	No.
2.00	0
4.00	92
6.00	110
8.00	0
10.00	0
12.00	0
14.00	0
16.00	0
20.00	0
26.00	0
>	0

Appendix 2 - Specification of M1 and M2.

```
//Specification of M1 and M2 processes which cooperate to
//perform end-to-end MSG measurements.

//M1 is the driving process for end-to-end MSG measurements.
//It begins by determining the parameters to be used for
//the measurement run.  These parameters include:
// HOST = host where M2 runs.
// SGMCNT = Number of times to repeat:
//      M1:                          M2:
//      SendGenericMessage      ->  ReceiveGenericMessage
//      ReceiveSpecificMessage <-  SendSpecificMessage
// SGMSIZ = Size of message in bytes for above measurement.
// SSMCNT = Number of times to repeat:
//      M1:                           M2:
//      SendSpecificMessage      ->  ReceiveSpecificMessage
//      ReceiveSpecificMessage  <- SendSpecificMessage
// SSMSIZ = Size of message in bytes for above measurement.
// SALCNT = Number of times to repeat:
//      M1:             M2:
//      SendAlarm  ->
//                 <-  SendAlarm
// OPBCNT = Number of times to repeat connection open measurement
//  for binary send (M1) to binary receive (M2) - 8 bit bytes.
// OPPCNT = Number of times to repeat connection open measurement
//  for binary pair - 8 bit bytes.
// OPTCNT = Number of times to repeat connection open measurement
//  for User TELNET (M1) to Server TELNET (M2).
// The connnection open measurement is:
//      M1:                   M2:
//      OpenConn          <->  Openconn
//      Send 100 bytes     ->
//      CloseConn         <->  CloseConn

M1:

query-user-for-measurement-parameters()
unless SGMCNT = 0 do              //Do send generic measurements.
{ Bytes(1-4,M) := 0               //0 bytes in message M signals
  until SGMCNT = 0 do             //M2 doing send generic measurement.
  { gather-start-data()
    SendGeneric(M2@HOST, M, SGMSIZ, ...)  //Send message M of size
    ReceiveSpecific()             //SGMSIZ generically to M2 and wait
    gather-stop-data()            //for reply.
    log-data()                    //Log round trip data.
    SGMCNT := SGMCNT - 1
  }
}
```

```
unless (SSMCNT=Ø)&(SALCNT=Ø)&(OPBCNT=Ø)&(OPPCNT=Ø(&OPTCNT=Ø) do
{                                   //Build mess holding measurement data.
 Bytes(1-4,M) := -1                 //-1 bytes identify this as meas data.
 Bytes(5-8,M) := SSMCNT             //Number of SendSpecific repetitions.
 Bytes(9-12,M) := SSMSIZ            //Size of specific messages.
 Bytes(13-16,M) := SALCNT           //Number of SendAlarm reps.
 Bytes(17-20,M) := OPBCNT           //Number of binary send/receive reps.
 Bytes(21-24,M) := OPPCNT           //Number of binary pair reps.
 Bytes(25-28,M) := OPTCNT           //Number of TELNET reps.

 SendGeneric(M2@HOST, M, ...)       //Send measurement data to M2.
 ReceiveSpecific(P, ...)            //Get ack from M2 - P = M2's name.

 Until SSMCNT = Ø do                //Do send specific measurement.
 { gather-start-data()
   SendSpecific(P, M, SSMSIZ, ...)  //Send specific message to M2.
   ReceiveSpecific(...)             //Read reply.
   gather-stop-data()
   log-data()                       //Log round trip data.
   SSMCNT := SSMCNT-1
 }

 unless SALCNT = Ø do
     AcceptAlarms()                 //Get ready to exchange alarms.
 until SALCNT = Ø do                //Do alarm part of expt.
 { EnableAlarm()
   gather-start-data()
   SendAlarm(P, SALCNT, ...)        //Send alarm to M2.
   wait-until-alarm-receipt-signaled()
   gather-stop-data()
   log-data()
   SALCNT := SALCNT -1
 }

 OPCON1(OPBCNT, binary-send-8)      //Do binary connection measurements.
 OPCON1(OPPCNT, binary-pair-8)      //Do binary pair measurements.
 OPCON1(OPTCNT, Server-TELNET)      //Do TELNET connection measurements.

}

save-or-display-collected-data()

StopMe
```

```
let OPCON1 (N, TYPE) =              //Routine to do M1's part for
{                                   //connections.
 until N = 0 do
 { gather-start-data()
   OpenConn(P, TYPE, N ...)         //Open connection to M2; type = TYPE,
   gather-stop-data()               //ID = N.
   log-connection-open-data()
   send 100 data bytes over connection
   gather-start-data()
   CloseConn(P, N, ...)             //Close the connection M2.
   gather-stop-data()
   log-connection-close-data()
   N := N-1
 }
}

//M2 is the responding process for MSG measurements.
//It merely "reflects" M1's actions.

M2:
AcceptAlarms()                      //Get ready to handle alarms.

ReceiveGeneric(P, M, ...)           //Receive message M from M1 process P.
if Bytes(1-4,M) = 0
then                                //This is part of send generic
{ SendSpecific(P, M, ...) }         //measurement.  Reply.
else                                //Generic message holds data for
{                                   //measurements.
 SSMCNT := Bytes(5-8,M)             //Extract number of specific messages...
 SSMSIZ := Bytes(9-12,M)            // ... size of specific messages
 SALCNT := Bytes(13-16,M)           // ... number of alarms
 OPBCNT := Bytes(17-20,M)           // ... number of binary receives.
 OPPCNT := Bytes(21-24,M)           // ... number of binary pairs.
 OPTCNT := Bytes(25-28,M)           // ... number of Server TELNETs.

 SendSpecific(P, ...)               //Reply to generic message.

 until SSMCNT = 0 do                //Do SendSpecifics.
 { ReceiveSpecific(P, M, ...)       //Read message from M1.
   SendSpecific(P, M, ...)          //and reply.
   SSMCNT := SSMCNT-1
 }

 until.ne 7
 SALCNT = 0 do          //Do alarm measurements.
 { EnableAlarm()
   wait-until-alarm-receipt-signaled()
   SendAlarm(P, SALCNT, ...)
   SALCNT := SALCNT-1
 }
```

```
    OPCON2(OPPCNT, binary-pair-8)   //Do binary pairs.
    OPCON2(OPTCNT, Server-TELNET)   //Do server TELNET

  }

  StopMe()


  let OPCON2 (N, TYPE) =            //M2's part of connection measurement.
  {
   until N = 0 do
   { OpenConn(P, TYPE, N, ...)      //Open connection to M1; type = TYPE,
     Read 100 bytes of data         //ID = N.
     CloseConn(P, N, ...)           //Close the connection.
     N := N-1
   }
  }
```

Appendix 3 - Sample MSG Event Log

The following is a portion of the event log for an OPEN operation. It begins with the execution of a SendGeneric primitive by a FM process and ends with execution of a SendSpecific primtive by a FLPKG process. The event log is followed by a description of the entries.

```
24:23.952  32412  21815   1595   9002 (425,11677,0) 9713
  3 FOREMAN.1006  (0.228 in 9.924)
   24:14.028  SendGeneric, Valid, Timeout=300000 UnBlock, PE  108
     Length=44 Handling=200 to WM
24:27.476  32642  22044   1595   9003 (425,11857,0) 9762
  3 FOREMAN.1006  (0.457 in 1:7.550)
    Completed MsgSend, Disp = Succeeded, PE  108
24:32.090  32925  22309   1613   9003 (448,12013,0) 9848
  3 FOREMAN.1006  (0.088 in 1.956)
   24:30.134  RcvSpecific, Valid, Timeout=900000 UnBlock, PE  109
24:32.114  60869  20961  23836  16072 (379,10565,0) 10017
  1 WM.1003  (0.149 in 4.302)
    Completed MsgRcv, Disp = Succeeded, PE  93
    Delay =10.622  Length =44 Handling =200 from FOREMAN.1006
25:13.994  65601  21372  28157  16072 (382,10718,0) 10272
  1 WM.1003  (0.209 in 11.165)
   25:2.829  SendGeneric, Valid, Timeout=600000 UnBlock, PE  110
     Length=129 Handling=200 to FLPKG
25:29.016    524    524      0      0 (0,5,0) 519
  Creating Process FLPKG.1011
25:35.449  65855  21619  28164  16072 (382,10875,0) 10362
  1 WM.1003  (0.049 in 1.297)
    Completed MsgSend, Disp = Succeeded, PE  110
25:35.933   1052    824    229     -1 (33,167,0) 624
  2 FLPKG.1011  (0.070 in 1.248)
   25:34.685  WhoAmI, Valid
25:39.008  66094  21824  28197  16073 (384,11007,0) 10433
  1 WM.1003  (0.053 in 0.647)
   25:38.361  RcvSpecific, Valid, Timeout=600000 UnBlock, PE  111
25:40.489   1310   1035    276     -1 (52,312,0) 671
  2 FLPKG.1011  (0.050 in 0.607)
   25:39.882  RcvGeneric, Valid, Timeout=0 UnBlock, PE  112
25:45.056   1611   1335    276      0 (52,485,0) 798
  2 FLPKG.1011  (0.350 in 25.849)
    Completed MsgRcv, Disp = Succeeded, PE  112
    Delay =36.871  Length =129 Handling =200 from WM.1003
     Length=75 Handling=0 to WM.1003
```

Each log entry has the same first line:

TimeSinceStartup Job MSG Process Delta (RUP,LOG,Status) MSGReal

Job:	Total Job CPU time.
MSG:	Total CPU time for MSG fork.
Process:	Total CPU time for top fork of involved process (if any)
Delta:	Job-MSG-Process
RUP:	Total CPU time used in routines that manage Recent User Primitives.
LOG:	Total CPU time used in routines that do event logging.
Status:	Total CPU time used in status reporting routines (S).
MSGReal:	MSG-RUP-LOG-Status: CPU time used by MSG fork for MSG functions.

The rest of the lines depend upon the type of event being logged. Job and Process Creations and Terminations should be self explanitory except that "All Forks CPU" is an estimate of total CPU time used by all forks of the process based on taking the total Job CPU time minus the MSG fork CPU time.

For user primitives the next line is:

JCB ProcessName (DeltaCPU in DeltaTime)

DeltaCPU:	MSG CPU time used since primitive was issued.
DeltaTime:	MSG real time since primitive was issued.

The succeeding lines should be self explanitory.

For Completed PEs the second line is the same as for User Primitives except that the Deltas are since the last receipt of an intra-Host MSG-MSG signal; they are thus not always of relevance. The other lines are self explanitory except that "Delay" is the real time delay from when the message (or alarm or connection) control block was created until it was delivered.